N92-16576

AUTOMATED PREDICTIVE DIAGNOSIS (APD): A THREE TIERED SHELL FOR BUILDING EXPERT SYSTEMS FOR AUTOMATED PREDICTIONS AND DECISION MAKING

Michael Steib
Vitro Corporation

Abstract. The APD software features include: On-line help, Three-level architecture, (Logic environment, Setup/Application environment, Data environment), Explanation capability, and File handling. The kinds of experimentation and record keeping that leads to effective expert systems is facilitated by: a) a library of inferencing modules (in the logic environment), b) An explanation capability which reveals logic strategies to users, c) Automated file naming conventions, d) An information retrieval system, e) On-line help. These aid with effective use of knowledge, debugging and experimentation. Since the APD software anticipates the logical rules becoming complicated, it is imbedded in a production system language (CLIPS) to insure the full power of the production system paradigm of CLIPS and availability of the procedural language C. This paper discusses the development of the APD software and three example applications: toy, experimental, and operational prototype for submarine maintenance predictions.

INTRODUCTION: FROM SHELL TO TOY TO OPERATIONAL PROTOTYPE

This paper describes the Automated Predictive Diagnosis (APD) environment for development of expert systems and discusses three example applications. It is for readers familiar with production system programming and the development process for building expert systems. After a preliminary look at the output of an operational prototype, the paper begins the discussion with a presentation of the assumptions on which the development was based and an overview of the Automated Predictive Diagnosis software. Once its features and how it is used are described, the architecture of the APD environment and its filing system are explained. This explanation is followed by a recount of the development from the APD shell to Toy Application to an Operational Prototype based on an Experimental Environment with real data. The Operational Prototype for Submarine Maintenance generates printouts of the following type:

```
SHIP: shp    REFIT: rf                                  NAME: name
                                                        DATE: date

name of part 1
VALUE: value1  THRESHOLD: threshold1
***SAT***                        UNTIL: mm-yyyy

name of part 2
VALUE: value2  THRESHOLD: threshold2
               ***UNSAT***
```

... and so on for the parts of the steering and diving of the submarine. The command to run the computer program and shp, rf, name, and date from the heading on the top two lines are all that is input from the terminal. Of these, only the refit number and the ship number are used by the processing of the software. The sat unsat designations relate to satisfactory and unsatisfactory status of the respective parts of the ship, shp, at the date determined by rf. The latter two, name

and date, are used only for identification. Computer files support the rest of the information: derivations to obtain trendable data from sets of untrendable data, parameters needed to run the APD software and predictions. These files contain constant defaults and updated measurement information. The "UNTIL" field is left blank in cases where no prediction can be estimated as to when trendable values derived from data sets would cross the threshold.

A demonstration at this level would consist of a command to start the program, entering the four, shp, rf, name, and date, letting the software run and taking the printout from the printer. In the following discussion, APD refers to the environment to build The APD First Toy Application, The APD Experimental Environment for Submarine Predictions, and The APD Operational Prototype for Submarine Maintenance. It begins with the assumptions on how APD is to support the standard evolutionary incremental build strategy for developing expert systems.

PHILOSOPHY OF THE SOFTWARE: BASIC ASSUMPTIONS WITHIN AN EVOLUTIONARY PROCESS MODEL

The philosophy of the APD software stems from a desire to support standard expert system development procedures like the Evolutionary Spiral Model being formalized by the Software Productivity Consortium, Herndon, VA . The APD concept is based on the following assumptions:

1) Unbundled software capabilities are becoming more readily available, so that the APD software should leave graphic user interface or data base management capabilities to other packages with open interfaces.
2) APD is to be used for expert systems projects where an incremental, evolutionary development process is needed. There will be emphasis on risk management and need for repeated experimental runs with changes in logic, parameters, and data. A standard knowledge acquisition process for expert systems is assumed. 2a) Incremental testing of expert systems should be facilitated. 2b) Software maintenance to update and improve the expert systems is needed. 2c) Programming language capabilities and the production system paradigm are important.
3) The ability to run on 640K is needed.
4) Performance time is not a constraint. In the Operational Prototype for Submarine Maintenance saving operator time is emphasized while saving computer time is not. In the Experimental Environment, ease of maintenance, automated filing, modularity, computer memory, minimal input from a terminal and the ability to scale up were given higher priority than performance time. APD supported this. Of course, in some situations, fast performance saves operator time.
5) Reusable modules are an advantage. A judicious choice of logic modules coupled with an application specific choice of setup parameters should result in the creation of new applications by non-programmers.
6) Standardized utilities across logic modules is an advantage.
7) Predictive diagnosis has priority over other expert system application areas.

The state of the art in expert system shells leaves many of the above to the programmer and often emphasizes coordination with other software (all too often interfaces with particular software as opposed to open interface), or quick prototyping. APD supports these indirectly. Programming power is derived from CLIPS and C, coordination with other software packages is left to open interfaces and portability of the C language, and quick prototyping is supported by the CLIPS production system paradigm and the APD reusable logic and setup modules. To date, no English like user interface has been added to the front of APD. Part of the rational for APD is the desire to support the rigors of the standard expert system development process rather than relegate development to application-programming environments that do not require it. An overview of the APD shell elucidates the development path from the assumptions to the printout of the introduction.

THE APD PURPOSE: AUTOMATED INFERENCING/DIAGNOSIS

The general purpose of the APD software is to automate inferencing processes. The goal is software that automates the process of deriving conclusions from assumptions/data. The APD environment supports experimenting with the process and improving domain assumptions. The software also facilitates record-keeping on domain assumptions and conclusions related to different data sets. More specifically, the purpose is to help automate the process of developing/deriving predictions, especially those related to maintenance needs, from diagnostic domain data and information. The APD software incorporates a methodology that facilitates modularization, a tiered sharing of setup and logic modules, helpful utilities shared across applications, and levels of transparency. Users and programmers may work with changes to the software on three main levels: (1) programming changes (minimum transparency), (2) leaving the programs as is and editing setup files to alter program behavior, and (3) answering queries from front end programs that automatically change these setup files (maximum transparency).

TECHNICAL APPROACH: ROBUST, YET USER-HELPFUL

The APD software is designed to facilitate the development of software systems for real-world situations where the logical rules governing the situation may become complicated by exceptions, complex interrelationships, uncertainty, and sometimes differing ideas and opinions. These kinds of complications are expected under assumption 2 above and suggest the following characteristics of the APD software:

1) The logical rules are embedded in the production system paradigm of the CLIPS language so that:

- a) The full power of the production system (rule based) language plus a conventional procedural language, C, is available,
- b) Porting to an even more powerful language (Inference Corporation's ART) is facilitated,
- c) Source code in C for the production system language is available.

These advantages help to handle possible complex relationships and exceptions to logical rules.

2) Features that facilitate the experimentation suggested by assumption 2, the convenience of assumption 6 and the reusability of assumption 5 include:

- a) APD shares reasoning techniques between applications via a library of inferencing modules,
- b) An explanation capability reveals reasoning strategies to users,
- c) Automated file naming conventions associate assumptions, data, and conclusions via file names,
- d) An information retrieval system for data, assumptions, and logic responds to user requests,
- e) On-line help is available via a menu system.

These features aid in debugging and possible indecisions and misunderstandings concerning characteristics of domain logic and data. They facilitate the kinds of experimenting and record-keeping that lead to resolutions via the empirical evidence from instantiations of special cases. The remaining assumptions (1, 3, 4, 7) which lead to the coordination with the unbundled approach to software, ability to run on 640K of memory and favoring the expert system application

area of predictive diagnosis are supported by the design details and implementation of APD and applications. Thus the APD software uses the off-the-shelf capabilities of CLIPS and facilitates repeated additions as the logic, setup, and data modules are built and saved. The CLIPS Help Facility is used to furnish on-line help for the APD software, its organization, methodology, and capabilities, as well as for the CLIPS software.

USE: CREATING MODULES AND RUNNING APD

There are two major steps in using the APD software. Step 1 is preparing the modules for a particular application area and application situation. Step 1 requires the least amount of work when the needed logic, setup, and data modules already exist for the application situation. If the data module does not exist, then it must be created. This may be done by:

1) Using an existing data module that works with the setup and logic, and
2) Changing the data element values, manually or programmatically.

Similarly, if the setup module for the application area does not exist, it can be created using an existing setup module that works with the logic.

Finally, the most work is required if none of the existing logic modules can be set up to serve the application. Creation of the logic module often requires more than changing another logic module template. It could require a production system programming effort to either alter another logic module or develop one using logical rules gleaned from a knowledge acquisition process. These jobs require a programmer and domain expert. Developing the logic module in an abstract form for use with other applications, although labor-intensive, helps increase the usefulness of the APD software. In the above cases, certain syntax requirements must be followed to ensure coordination with the APD software features: explanation capabilities, data and information retrieval capabilities, and file handling capabilities.

Step 2 is running the software with the appropriate logic, setup, and data to automatically derive conclusions. The files created in step 1 are loaded into CLIPS with APD software, and run to generate the files that contain the conclusions along with files that contain the information and data for automated explanation and data retrieval capabilities. User queries and menu choices can be used. The CLIPS Help Facility can be used to furnish APD as well as CLIPS on-line help.

COMPARISON TO OTHER SOFTWARE: PRODUCTION SYSTEM POWER

Although APD software capabilities may be developed and programmed in several programming paradigms (including functional, procedural, and object-oriented languages), APD was developed and coded in the production system paradigm of the language, CLIPS, to ensure that the full power of CLIPS is available. This CLIPS availability gives the APD software an advantage over off-the-shelf products. Of course, this advantage is not needed unless the complexity of an application requires it. Off the shelf products often provide English like user interfaces that require only minimal programming efforts. However, the cost of continuing past the simplistic environment of the first quick prototype may include an expensive customizing effort or redo. With the source code of the off-the-shelf product unavailable and yet the methodology already in place, the first prototype may need to be abandoned in favor of a more robust programming environment like that furnished via the APD/CLIPS environment.

ARCHITECTURE: INFERENCING MODULES AND UTILITIES

The APD inferencing software is organized into three major sets of modules: logic, setup, and data. The modules in the logic set provide general logic structures used for differing applications. Each module in setup tailors a logic module for a specific application area. Each module in data furnishes the measurements for a specific situation in the application area. Thus a logic module could be used with several application areas, each characterized by a setup module. Each setup module could be used with several situations each characterized by a data module.
Program modules that are useful across application areas and situations relate to filing, file naming conventions, on-line help, query capabilities, explanation capabilities, and data handling. These features as well as the design of the APD software support the expert system development process. The programs and files of APD are in the following directory structure:

The directory APD is the top directory. It contains the other directories.
The addhelp directory contains help files to furnish information to users on CLIPS and APD.
The data directory contains the data files.
The help directory contains programs and utility files to help manipulate the environment where APD is run.
The loaders directory contains programs and files to determine the selection of which data, setup and logic files are included in a run.

> There are two methods suggested for running a choice of modules. One uses loader files which specify modules in the APD system to load into CLIPS and run. The other uses a program which loads the modules of a user specified list. In this method, the lists are all in one file, a list of lists. In each of these methods there is the opportunity to run modules in small enough sets to accommodate the 640K memory constraint of DOS. Indeed in the Operational Prototype a batch file does this with the second method. The batch reuses the loading program and takes the place of the user to set up the software sequentially, part after part.
>
> The directory, loaders, contains a file to tell APD which type computer is being used. Some of the development was done on the Macintosh, some on the IBM PC AT. The filing software was made general and instantiated to the particular operating system. Except for this, the filing system is the same on the two platforms.

The logic directory contains the logic program modules.

> A simple example of a bit of logic is that a flag is set if a data element value crosses a threshold. A setup file then tells which data element and what value to use for the threshold check. In the following a detailed example of logic is given in the Toy Application.

The programs directory contains the programs for explanation capabilities, retrieval capabilities, comment capture, and a filing system.

For each of the following functions, there is a module in the programs directory:

Prepare the data.
Extract the data from prepared data bases.
Derive the trendable data in the Experimental Environment and Operational Prototype.
Automatically alter the setup files to operator specification.
Automatically reinitialize the setup files for reuse.
Translate the results for display to operator.
File the information for query response and explanations.

Save operator comments.
Display requested results and information.
Explain the results of a run.
Provide a menu for choosing desired information retrieval.
The menu options are chosen with numbers or first letters:

1 (show data elements)
2 (show data value)
3 (show groups)
4 (show result explanation)
5 (show results)
6 (show satisfied criteria)
7 (show satisfied groups)
8 (show tests)
9 (show test criteria)

Explain certain CLIPS error messages for expected user errors.
Help coordinate between APD modules.

The results directory contains the directories, outputs and comments.
Outputs contains the logical results of runs.
Comments contains the user comments.
Comments is usually used only during experimental and test runs.
The setup directory contains the files that are used to instantiate the logic programs.
For instance, if a logic program sets a flag when the value of some data element crosses a threshold, then the setup file instantiates the logic with a data element and value for the threshold.

When automated filing is chosen, it uses this directory structure to organize the run results: outputs and comments stored in the results directory.

The design of the APD software supports modularization and layering. The modularization is implemented with the division of the software into data files, setup files, logic programs, and utility programs. This modularization is augmented by layering. Layering relates to the level of transparency with respect to details of functionality. For the APD software there are three main levels of transparency. These three are variations on instantiating the core logic programs. The first instantiation level is the programming which creates the logic programs. These programs comply with formats needed to use the APD utilities. The programming effort requires enough knowledge of how the logic programs work to yields general programs. They are instantiated on a second level by setup files and data files. This requires less knowledge about the details of APD implementation. However, this editing still requires knowing format requirements for the APD files. The third level of transparency is furnished by programs which automatically insert the setup and data information. At this third level a user answers queries on what to use in the setup and data files and does not need to be concerned with the formats of the files since the input into these files is automated.

These three levels have the customary property: each layer needs to interface with only the next layer down. So to write new programs to query for the setup and data information, the formats for these files must be understood, but not the implementation of the logic programs. Using modularization and layering in this context furnishes an environment for reusable modules, standardized utilities and accommodation of a 640K memory constraint.

THE FILING SYSTEM: AUTOMATED DEVICE FOR ASSOCIATING INPUTS AND OUTPUTS

The APD filing has conventions for directory structure and file names in DOS. The diagnostic logic, setup and data files have common name extensions. The setup files are named nnnapdst.eee where nnn is chosen by the user (a number between 001 and 999 is often used) and eee is the file name extension of the logic file. Similarly, the data file is named nnnapdat.eee, where nnn is chosen by the user (a number between 001 and 999 is often used) and eee is the file name extension of the logic file.

Other files have a name that uses the first two characters to designate what type of file they are, the next three characters to designate the associated setup file, the next three characters to designate the associated data file, and finally, the extension of the logic file. For instance, if the setup file is 001apdst.d05, the data file is 002apdat.d05, and the logic file has file name extension d05, the facts file created by APD would have name ft001002.d05. The user is given the option of not using these filing conventions. The Software queries for either user supplied names or the OK to use automated naming for the results files. For the comments files, the operator supplies the names and depends on the APD software to capture comments and "process location" of comments.

DEVELOPMENT EXPERIENCE: A SEQUENCE OF PROTOTYPES FROM TOY TO OPERATIONAL

The development of the APD software led to an Operational Prototype application for day to day use in predicting submarine maintenance requirements. It began with the implementation of a preliminary shell for development of experimental environments. This was tested with the Toy Application Program. Then the prototype Experimental Environment for predicting submarine maintenance requirements for steering and diving parts was developed. Finally the modules from the Experimental Environment were used to develop a day to day Operational Prototype. This prototype assesses the steering and diving parts, ship by ship, as they arrive and generates a printout like in the introduction to this paper. The following discusses the development of these applications.

REQUIREMENTS: DATA, SETUP, LOGIC, AND PROGRAMS

The programs that furnish a foundation of utilities for APD applications require a compliance to format in the three APD modules: data, setup, and logic. So the programs and files of the three modules are implemented with adherence to the APD formats as well as CLIPS syntax. The following are the example applications built in CLIPS within the APD environment.

THE TOY APPLICATION: BASIC TEST AND DEMONSTRATION OF THE APD SOFTWARE

The first example application was a toy program. In this program data was entered at a terminal since the data base size was manageable. The program was run with test data and the APD shell was debugged. This was accomplished with a domain expert other than end users. However the development of the Experimental Environment and Prototype Application used the target operators as domain experts. The Toy environment depends on checks to see if values exceed thresholds. If a sufficient number of the measurements do exceed the related thresholds, then a "check group" of such threshold checks is called satisfied. When this happens, a postponement of maintenance is

recommended. The following presentation of the toy logic is designed to be readable, yet somewhat like the implementation code. The ?'s signify variables.

SAMPLE RULES IN THE COMPROMISE LANGUAGE FOR TOY LOGIC ENVIRONMENT

DEFER_MAINTENANCE_1 Suggests postponing maintenance

IF
1. lead time for ?x is ?lead_time for ?maintenance planning
2. ?x is scheduled for next ?maintenance at time ?maintenance_time
3. present time is ?present_time with ?present_time < ?maintenance_time -?lead_time
4.result number ?n - ?x at time ?present_time has ?maintenance status not needed for a time interval of ?time_till_maintenance_is_needed
5. ?x has ?maintenance scheduled at intervals of length ?time_between_maintenance_performance

THEN
IF
the time interval ?time_till_maintenance_is_needed > (?maintenance_time - ?present_time) + ?time_between_maintenance_performance
ASSERT
result number ___ - at time ?present_time the condition of ?x indicates that maintenance ?maintenance should be postponed at least until the regularly scheduled maintenance time for ?maintenance after this next one derives from result number ?n
AND
IF
the time interval ?time_till_maintenance_is_needed > (?maintenance_time - ?present_time)
ASSERT
result number ___ - at time ?present_time the condition of ?x indicates that maintenance ?maintenance may be postponed until (?present_time + ?time_till_maintenance_is_needed) derives from result number ?n
END OF RULE

INCREMENT COUNTER FOR SATISFIED CRITERIA Increments counter by 1 for each time the criterion for a check in a check group is satisfied

IF
1. result number ___ - ?x has check counter ?check_counter for check group ?check_group at time ?present_time
2. for ?x the check ?check_test in check group ?check_group at time ?present_time satisfied the criterion

THEN
ASSERT
result number ___ - ?x has check counter (?check_counter + 1) for check group ?check_group at time ?present_time
END OF RULE

QUALIFICATION FOR A TIME PERIOD BEFORE MAINTENANCE Notes status of an item in the case when there is satisfaction of criteria for all the checks in a group which is associated with a time period before maintenance is needed.

IF
1. the number of check tests for ?x in check group ?check_group is ?number_of_checks
2. result number ?n - ?x has check counter ?number_of_checks for check group ?check_group at time ?present_time
3. for ?x the check group ?check_group is associated with maintenance ?maintenance and with postponement time interval ?time_till_maintenance_is_needed

THEN
ASSERT
result number ___ - ?x at time ?present_time has ?maintenance status not needed for a time interval of ?time_till_maintenance_is_needed derives from result number ?n
END OF RULE

SIMPLE ABOVE THRESHOLD Notes satisfaction of criterion that a data element has value above a threshold.

IF
1. ?check_test is a check in the check group ?check_group for ?x
2. the criterion for ?check_test is that data element ?data is greater than the threshold ?threshold
3. the data element ?data > ?threshold at time ?present_time

THEN
ASSERT
for ?x the check ?check_test in check group ?check_group at time ?present_time satisfied the criterion
END OF RULE

The following are example setup and data to be used in running the toy example application.

;;EXAMPLE SETUP INPUT

(deffacts example-setup "sets up the software for the checks and groups of checks to be made"
(the setup file is "001apdst.d05") (lead time for part_1 is 2 for replace_bushings planning)
(part_1 has replace_bushings scheduled at intervals of length 20)
(the number of check tests for part_1 in check group check_bushings is 3)
(result number -1 part_1 has check counter 0 for check group check_bushings at month apd_date 0 derives from counting satisfied criteria)
(for part_1 the check group check_bushings is associated with maintenance replace_bushings and with postponement time interval 40)
(test_1 is a check in the check group check_bushings for part_1)
(the criterion for test_1 is that data element measurement_1 is greater than the threshold 1)
(test_2 is a check in the check group check_bushings for part_1)
(the criterion for test_2 is that data element measurement_2 is greater than the threshold 3)
(test_3 is a check in the check group check_bushings for part_1)
(the criterion for test_3 is that data element measurement_3 is greater than the threshold 1))

;;EXAMPLE DATA INPUT

(deffacts example-data "data to be checked"
(the data file is "001apdat.d05")
(part_1 is scheduled for next replace_bushings at month apd_date 5)
(the present is at apd_date 0)
(the data element measurement_1 has value 2 at month apd_date 0)

(the data element measurement_2 has value 4 at month apd_date 0)
(the data element measurement_3 has value 2 at month apd_date 0))

EXPERIMENTAL ENVIRONMENT FOR PREDICTIONS: HANDLING REAL DATA

The second example application was with real submarine measurements taken at the steering and diving parts and stored by computer. Availability of this computer readable information and the larger size of the data set suggested a departure from the terminal input of the Toy example. The APD Experimental Environment included programs to automatically create a data input file from the computer readable measurements, and a C program to format the data for use by the APD prototype software. Since the prototype required that the data input file receive trendable data, the software was designed to transform the sets of measurements to trendable derived data. The trends of the derived data were used to determine predictions on when predetermined thresholds were crossed. These threshold crossings were the desired outputs. So, the success of the Experimental Environment for Submarine Maintenance Predictions depended on:

(1) deriving trendable data elements and values,
(2) the trending methods, and
(3) the threshold values. All three of these were supplied by users who served as domain experts. The system was implemented with:
(1) data preparation software in the "C" language to create files that facilitated APD extraction,
(2) setup files to determine how the data extractions and derivations were made,
(3) data extraction and derivation programs that extracted the data from the prepared source files and derived the data elements for trending,
(4) data files for storing the derived data
(5) setup files to determine thresholds and how the trending was to be done,
(6) logic programs that did the trends and compared values with thresholds,
(7) Programs that saved and explained results.

Note that the setup files of (2) and (5) furnished convenient means of changing the way that the extraction and derivations of (3) were made and the way that the trending and comparisons of (6) were done. These setup files furnish convenient generality, a way of using (3) and (6) in different application environments. The choices made available by the setup for extraction and derivation include:

the list of ships to be processed,
constraints on what data is acceptable,
designation of file to store information on inappropriate data,
location of data elements in the lines of the original files,
lines of original file to be processed,
years to be processed,
where to find dates in the original data file,
designation of original file of data,
designation of storage file for derived data,
names of the data elements,
configuration of data use by derivation program,
names of data to be used in result files.

The choices made available by the setup for the trend logic include:

group result names (or a "not used" designation),
names of tests made in the data groups,
criteria names for the checks in data groups,
parameters for estimating last maintenance actions,
parameters for appearance of printouts,
names for files produced,
type of trending.

In runs through the submarine maintenance measurements where there was enough data to yield a sequence of the derived data over time, trends were made. Predictions on timing of maintenance requirements were presented in the form of:

(1) pictures fashioned for ascii character printouts,
(2) tables to be used by graphics packages and
(3) information to be used by the APD utilities.

The utility programs, in turn, responded to user queries on what data values, setup parameters, and logical inferencing was used to get the predictions. The following files were generated by the APD software:

(1) the derived data files,
(2) the "unusable data" files and
(3) the setup files with parameters of the run.

For later reference, data, setup, logic and predictions were associated via file naming conventions for debugging. This will also be useful if and when improvements or updates to the domain assumptions are considered. The main part of these assumptions relate to selection of derivations and trend logic to apply to the different data sets. Since the original data elements were not trendable in a meaningful way, they were grouped into sets which led to files of derived data which was trendable. Creation of the setup files, both for extraction and derivation and for the trend logic was done via editing template setup files. The output files were handled by the automated filing system (except when capture of operator choice of file names was being tested). The Operational Prototype is a sequence of special cases of the software in the Experimental Environment and the following section on this Prototype augments this discussion.

OPERATIONAL PROTOTYPE FOR SUBMARINE MAINTENANCE: A BATCH CONFIGURATION

The third example application of the APD system uses the modules of the second example in a batch mode and assumes all setups constant in the sense that they are not changed as in experimentation with different parameters and logic. These setups do vary from part to part, automatically. When in the batch the user is not bothered with choices. The batch program is set up for the routine job of deciding where to apply maintenance resources for ships when they arrive. It uses a setup which asks the user to stipulate only the ship and date. For this date and for each part, the system returns either unsat or sat depending on whether the derived value has crossed the threshold or not. A special module to present this information part by part for this special status date was added to the Operational Prototype. (The experimental environment presents the date when a value crosses a threshold rather than the status at a "special status date.")

The Operational Prototype batch run does the following:

It erases the output files from previous runs.
It erases the data files from previous runs.
It reinitializes the setup files for data extraction and derivation of trendable data.
It prepares the data for the APD Operational Prototype.
It queries the user for the ship, special status date, user name and date of the run and files this information to be sent to the printer.
It edits the setup files to include the information on ship and special status date.
For each part:
    It selects the data derivation for the part.
    It selects the extraction and derivation setup file.
    It selects the trending setup file.
    It makes the prediction on when the derived data is expected to cross threshold.
    It files the results in the results directory and generates the printout.

Some thresholds are lower thresholds, some are upper thresholds. The trends are based on trendable values derived from the data elements extracted from the data bases. The different data element values are date dependent. A date is considered a valid date only if all the data elements required to derive a trendable data value for that date are found in the data base. The predictions, along with the derived data values at all the valid dates are filed. The predictions are saved for the printout and the values at valid dates are saved for inclusion in visual explanations of the predictions. Also the program for comparison of derived values and threshold at the special status date is included in the run and the resulting sat-unsat status at the special status date is filed for printout. This is done for each special status date that is valid. The file for saving information for the printout is considered complete when the above has been executed for all parts. The input to the batch run consists of four pieces of information. They are the (1) ship and (2) special status date for stipulations of the setup files (3) users name and (4) date of the run. The latter two are for identification on the printout. The ship and special status date are constant throughout the batch run. The output is a list of the parts, the status, sat or unsat, and for each part marked satisfactory, a prediction as to when the derived value for the part will cross threshold (unless there is not sufficient data).

CONCLUSION: APD FACILITATES THE EXPERT SYSTEM DEVELOPMENT PROCESS

The APD software furnishes automated capabilities which aid in the execution of the evolutionary process of expert system development. Storage of information that characterizes experimental runs, retrieval of information on runs, explanation of the inferencing, modular organization, and levels of transparency are emphasized to help make accepted expert system practices convenient without losing the power to work with the complexity of real world problems. The APD shell was used with three main prototypes: a Toy Application, an Experimental Environment with real data, and an Operational Prototype. On-line help was coordinated with the CLIPS help facility. The software was developed on an IBM PC AT, a Zenith Laptop IBM AT compatible, and a Macintosh. Its modularization supports running on only 640K of memory by running small enough groups of modules, one after another. In the Operational Prototype, this was done in batch mode.

REFERENCES

Boehm, B. (1988). A Spiral Model of Software Development and Enhancement, *Computer,* May: 61-72.

Software Productivity Consortium. (1991). *Evolutionary Spiral Process Guidebook,* SPC-91076-MC. Herndon, Virginia